OPTIMIZING CONGESTION CONTROL BY USING DEVICES AUTHENTICATION IN SOFTWARE-DEFINED NETWORKS

Tamer Barakat, Hanan Eljawhri, Mohamed Merzban and Mahmoud Elbayoumi

Department of Electrical Engineering, Fayoum University, Egypt

ABSTRACT

The Internet and local networks (LAN) are essential in all organizations and aspects of our lives. These networks' performance should be at high speeds to perform efficiently. This thesis suggests several motions to improve performance. The first is a software-defined network (SDN) distinguished from traditional networks by its ability to program traffic, agility, and support for applications that need big data and virtualization. The second is to control congestion by rerouting traffic to the shortest path. Finally, to modify the above, device authentication reduces congestion and improves network performance.

KEYWORDS

Congestion Control, Devices Authentication, Rerouting, Shortest Path, Software-Defined Networks

1. INTRODUCTION

Every enterprise should consider the advantages and disadvantages of different types of networks for creating its networks. Consumer requirements for performance and flexibility, the demand for modern networks helped in SDN's ascent, and the increasing use of SDN and other virtualization systems. Traditional networks can not keep up with the ever-growing needs of users in the modern workplace. Users who want to scale their network infrastructure with as little downtime as possible are turning to SDN. It consists of the control plane and the data plane. It makes networks programmable and manageable. The benefits of the split are flexibility and cost savings in both capital and operational expenses.

Because of its many advantages over traditional networks as improved automation-based reliability, more effective network management, cost savings, and quicker scaling, it is software-based; but traditional networking is primarily hardware-based. Whereas SDN is software-based, it is more flexible and adaptable to users to manage resources virtually via the control plane [1].

The SDN has an application plane, a control plane, and a data plane. The SDN application layer is a set of programs; responsible for applications or services on a network and describes the service performance of network resources programmatically. These programs connect with the controller (control plane) through the northbound interface (NBI), and the controller modifies the activities of network resources; presented to it through the application layer programs for internal decision-making [2].

In SDN, the control plane (controller) works as a network operating system or the brain of the network because of its role in managing and configuring the devices by installing appropriate flow tables that contain packet routing rules on the network devices (switches, routers) at the data layer

out of the southbound interface (OpenFlow), which allows the SDN controller to add, modify, and delete flows in the flow table of an OpenFlow switches (data plane) and connects the controller to an OpenFlow switches to manage devices, receive packets from the OpenFlow switches, and send packets via the controller [3].

SDN controllers have many types; NOX was the first Open Flow compatible SDN controller. It uses C++. POX uses Python; it only supports OpenFlow version 1.0 and does not operate in a distributed way. Ryu is another Python-based SDN controller. In contrast to POX, it supports the OpenFlow protocol in multiple versions: 1.0, 1.2, 1.3, and 1.4. Additionally, it bolsters other protocols like Netconf and OFconfig. It has the benefit of being able to operate in a dispersed way [4].

The data plane consists of network hardware such as switches, routers, and packet routing tables. The network device that accepts packets through its ports can carry out network operations since packets can be forwarded or destroyed or have their header changed for a particular action after being received on ports. According to the packets sent by the switch upon a switch request (table-miss), the controller modifies the forwarding table for each switch request. It then delivers a response via a packet-out or flowmod message and keeps track of all application requests [5].

Dynamic and unstable network traffic is present. Network performance will drastically decline if a subnet has excessive data packets. The main reason for packet loss is network congestion, which happens when the network load exceeds the network capacity. This is why there is an urgent need for congestion control algorithms. Congestion control is the process of developing a network control policy based on the status and characteristics of the network [6].

The algorithm of congestion control has three main categories: loss-based algorithms, are define congestion when buffers are filling, then packets are dropped; delay-based algorithms, which rely on RTT measurements and identify congestion by an increase in RTT, which indicates buffering; and hybrid algorithms, which combine elements of the first two categories [7].

Loss-based algorithms were the initial congestion control algorithms, with TCP Reno being the first to be widely adopted, then the updating of loss-based algorithms, such as NewReno. But, Packet loss occurs when several flows with different RTT values compete on the same network; the flows with lower RTT values consume more bandwidth than the others, and a user with longer RTT flows will not be able to receive sufficient bandwidth. So these modifications could not resolve the RTT-fairness problems [8], BIC (Binary Increase Congestion Control), and Hybla as solutions to this problem. The Slow Start and Congestion Avoidance phases of the NewReno update by Hybla and made somewhat independent of RTT. However, the attained RTT fairness resulted in more extreme behavior from flows with higher RTTs. BIC's key objective was to employ a binary search method to receive close to the ideal congestion window size. Later analyses, meanwhile, revealed that BIC can still have RTT fairness worse than Reno. In contrast to loss-based algorithms, delay-based algorithms are proactive [9].

When bottleneck queues happen, delay-based approaches reduce network congestion to provide lowlatency data transfer and convergence to a fair share amongst flows. Many delay-based algorithms provide high and steady throughput by minimizing variation in data sending [10].

To improve congestion control in SDN by excluding unauthorized devices from authentication due to the possibility of attackers among them. Also, unauthorized access makes a hacker add conflicting network regulations or change them. On the other hand, protocol errors, device impersonation, and assaults involving the insertion of false packets all can cause changes in network topology [11].

Flooding of the controller's switch table in SDN: The acceptance of OpenFlow control packets at the controller leaves attacks possible because there is no way of authenticating and verifying the identity of the sender of incoming packets. So, an attacker can save data about fictitious switches in the switch table of the target controller. Due to persistent flooding of its related switch table, continuous execution of this attack degrades controller performance [12]. The attacks and abnormal behavior, including eavesdropping, packet injection, packet crafting, traffic interception, network information poisoning, and identity theft, are made possible by plain text channels and improper or weak authentication mechanisms.

To implement a trust mechanism to authenticate and authorize network privileges to prevent anyone from connecting to the network. In addition to encryption techniques, only data protected and illegal network access stopped. Thus, SDN security is strong by utilizing authentication and trust methods to encryption techniques, especially at the control plane. By exchanging signed certificates, symmetric and asymmetric keys, or message authentication codes, the controller must be able to identify and then authorize trusted devices (other controllers, switches, and extra appliances) and network applications, degrades controller performance due to persistent flooding of its related switch table, continuous execution of this attack [13].

In this thesis, we first employed rerouting traffic to take the shortest route that has minimized utilized bandwidth possible away from the congestion path that has maximum utilization bandwidth to provide low latency and high throughput, second; we designed a device authentication program between the authentication server and users to check if these devices are trusted. The server has the file of the media access control address (MAC addresses) of authorized devices; therefore, it confirms that they are in this file and then sends that information to the controller to allow those devices to access the network or block and disconnect them in the case of unauthorized devices. This program improves control of congestion and increases network performance.

1.1. Background and Related Work

Various related tasks provide networks the ability to route traffic. In [14], a work by Sofia Naning et al., using comprehensive controller information, implemented a congestion control technique. Congestion control techniques; are implemented by integrating multipath routing techniques with rate adaptability. The controller uses data from switches to choose the shortest path between the source and destination and the correct flow rate. But they did not illustrate the topology used.

In [15] work by Renuga Kanagavelu et al. propose a local rerouting strategy in SDN-based data center networks. Based on our flow classification scheme, the rerouting approach will redirect flows (at the point of congestion or one hop before) to possible paths; they cared about the size of the transmitted data as a classification, regardless of its importance and priority in transmission.

In [16], Masoumeh Gholami et al. control the SDN data center congestion using the OpenFlow protocol. When a network link becomes congested, the OpenFlow-equipped switches' port statistics; are centrally monitored, and part of the flows is diverse through paths with more available resources in the link. However, the average delay was reduced only crucial slightly in the last period of the test.

Umme Zakia et al. [17] used a dynamic load management strategy based on SDNs to maximize link utilization in Data Center Networks (DCNs) for taking flow priority into account. The program estimates the cost of each link and determines the shortest routes between each host and the other hosts. When a path becomes congested, its optimal alternative with the lowest link cost and the lowest traffic flow takes its place. Throughput, latency, and packet loss in a DCN have assessed

the algorithm's performance. But did not been specified when congestion may occur on the link to try to avoid or reduce it.

As for the authentication of SDN, there are related works like [18] by Diogo Menezes et al., Introducing AuthFlow, an access control and authentication system based on host credentials, the framework for control applications that enables software-defined network controllers to specify forwarding rules using the host identity as a new flow field. But they did not illustrate the topology used.

In [19], Jing Liu et al. designed a secure and reliable access method using SDN. The procedure involves a security access authentication mechanism and an access architecture design. The particular organization and implementation of data exchange during the access procedure are laid forth in the security access authentication protocol. The access device and the user's identification evaluate for credibility using the architecture and protocol. But the hardware environment consists of one access device, not more.

Our thesis first computes all the shortest paths between any two edges of an SDN network using Ryu controller [20], OpenFlow 1.3 [21] for the Ryu controller and Mininet [22], and secondly controls congestion by detecting the congested link with the lowest bandwidth, and then selecting the shortest path with higher bandwidth than the congested one to alter it of the flows. The proposal is to design an authentication protocol between client devices and the authentication server to prevent access to unauthorized devices by communicating the protocol with the controller, which increases network performance and reduces latency.

2. MOTIVATIONS AND CONTRIBUTIONS

2.1. Motivations

Security, access control, error minimization, connectivity, congestion control, traffic monitoring, and other issues are all relevant to SDN.

In this research, we concentrate on two problems of Software Defined Networking (SDN): congestion control and authentication. One of the main factors limiting SDN's performance is secure authentication. To illustrate, a high percentage of SDN controllers now in use lack a user access control function means that when the user and device access the network, there may be a problem with identity authentication.

This problem can result in forgery attacks or unauthorized users and devices of identity to illegally obtain information and cause congestion that lowers the quality of service and results in queuing delays, packet losses, and the blocking of new connections.

2.2. Contributions

The objectives of this thesis are achieved by designing an efficient and secure authentication protocol between access devices and an authentication server to achieve access authentication for devices across the entire network to improve the network performance and reduce latency and by designing an efficient congestion control model for the Ryu controller by centrally monitoring the statistics from OpenFlow switches, then rerouting flows in congested links through the shortest paths with free resources to improve the performance of SDN accordingly.

3. THE PROPOSED APPROACH

We use SDN, which has a structure illustrated in Figure 1. Ryu controller OpenFlow switches and hosts (devices). We will apply the rerouting approach for congestion control and then improve this with a secure protocol using the devices' authentication protocol.

Mininet generated the network topology diagram used in the proposal. The topology consists of a Ryu controller, 13 virtual switches (s_1-s_{13}), and 16 hosts (h_1-h_{16}) in Figure 2.

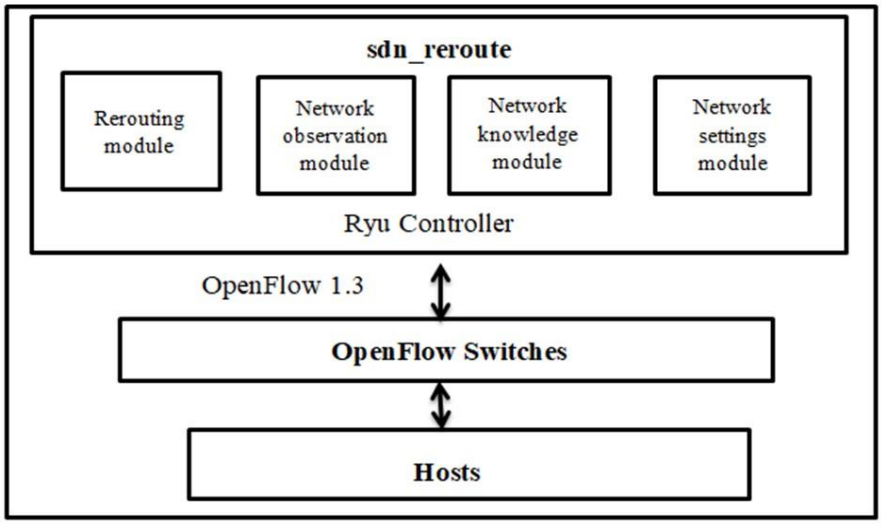

Figure 1. Structure of SDN

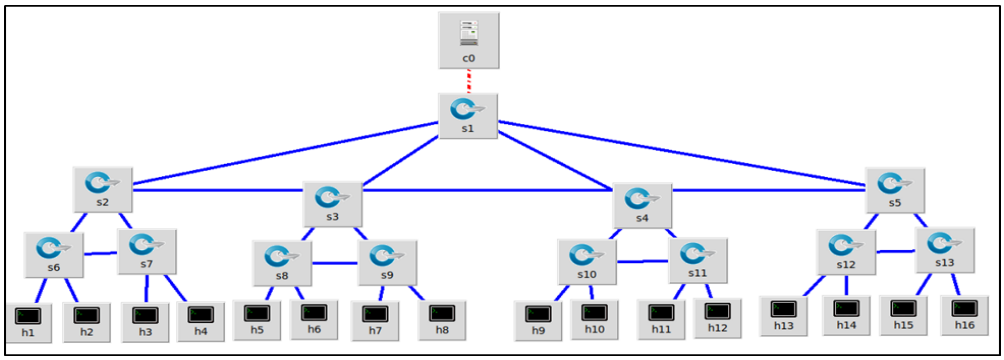

Figure 2. The network topology

3.1. The Rerouting Approach

In this approach, the SDN controller chooses the shortest path for routing traffic. High bandwidth utilization on links makes network congestion and lower network Quality of Service (QoS) guarantees possible. It is possible to successfully prevent network congestion by monitoring the underlying network data and controlling route forwarding to minimize forwarding delays.

The **sdn_reroute** application wrote in python language for the approach for the Ryu controller. This application consists of four modules: the **reroute** module, **monitor** module, network **knowledge** module, and network **settings** module, as illustrated in Figure 1.

In this application, the controller $C0$must obtain information about network resources or topology discovery, such as the number of switch nodes, topology of the network, and link resource information. Each OpenFlow switch must provide statistics to a controller, requested to OpenFlow switches to obtain the per-table, per-flow, and per-port statistics.

With this information and statistics from the network **knowledge** module, we used the $\boldsymbol{topology}$ library in this module as a result. Once the $\boldsymbol{topology}$ library has been imported into a module, it can be added using the $\boldsymbol{get\_topology\_data}$ function**,** which contains methods such as $\boldsymbol{get\_switch}$ for retrieving the list of switches and $\boldsymbol{get\_link}$ for retrieving the list of links [23].

Moreover, we used the $\boldsymbol{networkx}$ library in the same module. Once imported, the library in the module can add shortest-path algorithms that create a route between two nodes in a weighted graph such that the path's edge sum is reduced to a minimum. The weight is fixed and set to 1 for each link.

The procedure looks up the weights on the (outward) going (in weighted graphs) edges starting from the source node to the destination node. It selects a path and the shortest path's total [24].

Any link has parameters that the approach can employ, such as the fact that the remaining bandwidth of the link, $\boldsymbol{rem\_bw}$, at the start is equal to the bandwidth of each link, $\boldsymbol{link\_bw}$, and then it decreases continually by subtracting the amount of bandwidth required for the data flow, $\boldsymbol{flow\_bw}$, from $\boldsymbol{link_bw}$with each flow.

Additionally, the minimum bandwidth, $\boldsymbol{min\_bw}$, is initially equal to the maximum capacity of the link, $\boldsymbol{BW}$, and then it computes the minimum of the remaining bandwidths of the links, $\boldsymbol{rem\_bw}$ s . All of these parameters are predefined in the **sittings** module by their values, which indicate that $\boldsymbol{link\_bw}$ is set to $100\ Mbps$, $\boldsymbol{min\_bw}$ is set to $3\ Mbps$, and $\boldsymbol{BW}$ and $\boldsymbol{flow\_bw}$ are both set to $3\ Mbps$.

The main module is the **reroute** module, which obtains all paths between any two nodes; these paths are sorted from shortest to longest from the network **knowledge** module, the **reroute** module also obtains the shortest path between any two nodes as well as host information, whether source or destination, such as a MAC address and an Internet Protocol address (IP) from the network **knowledge** module.

Moreover, this module detects the congested link that has the least $\boldsymbol{link\_bw}$ as a result of consuming bandwidth by more flows, i.e., the $\boldsymbol{rem\_bw}$ of the congested link is less than the $\boldsymbol{BW}$ **or** the$\boldsymbol{min_bw}$ of a path is less than the $\boldsymbol{BW}$, and then it reroutes to the shortest path that has links that have a $\boldsymbol{rem\_bw}$ higher than the $\boldsymbol{BW}$.

The **monitor** module allows the controller to assess the amount of bandwidth needed for the data flow and track link bandwidth utilization, where the minimum bandwidth links,$\boldsymbol{min\_bw}$**,** and bandwidth remaining at edges, $\boldsymbol{rem\_bw}$**,** are computed during data flows for hosts.

To better illustrate the congestion control approach, the pseudocode is used in Algorithm 1, where $x, y, z,$ and m are nodes, **A** is the source host, and **B** is the destination host. Moreover, two examples in the Experimental Results and Analysis section will be mentioned later.

Algorithm 1 (reroute approach)

1- *Send CO request to OD switches to obtain statistics for each port, flow and Table*

2- The switches replay the requested statistics to Controller
3- receive all paths $\boldsymbol{Pth}$ and select the shortest path between any two nodes
4- At the start let $\boldsymbol{link\_bw = rem\_bw}$, $\boldsymbol{min\_bw = BW}$ and $\boldsymbol{flow\_bw} = 3\ Mbps$.
5- for each flow F has the shortest path $[x, y, z, m]$ from **A** to **B** do
6- compute $\boldsymbol{rem\_bw = link\_bw - flow\_bw}$ for each edge (link) in the path
7- compute $\boldsymbol{min\_bw} = min(\boldsymbol{rem\_bw}\ s\ for\ all\ edges\ in\ the\ path)$
8- if ($\boldsymbol{rem\_bw > BW}$ || $\boldsymbol{min\_bw > BW}$:)
9- print("no congestion")
10- end if
11- else if ($\boldsymbol{rem\_bw > BW}$ || $\boldsymbol{min\_bw > BW}$):
12- print ("**happen congestion**")
13- detect congestion link
14- select the shortest path that has $\boldsymbol{rem\_bws}$ of links $< \boldsymbol{BW}$ from $\boldsymbol{Pth}$
15- end else if
16- end for

3.2. The Authentication Approach

The flexibility and programmability of networks afforded by SDN are increasingly needed as the computing and communication industries evolve; however, security is one of the main factors limiting SDN progress. The majority of SDN controllers now in use lack user access control features. When a user or device accesses the network, authentication issues may arise.

This flaw can result in identity forgery, unauthorized access, and control of the entire network, which could result in congestion or even paralysis. In this thesis, we proposed a trusted access method for device authentication to address the issue of security checkups for access terminals. We created and implemented a software-defined network trusted access protocol or technique that might serve as the first network access defense to improve congestion control [25].

This technique is a device authentication protocol between clients and the authentication server in python language. The clients' devices to connect the network require a connection to the authentication server to secure the network against unauthorized devices. The server asks for the MAC of the devices from the clients to verify that they are authorized. If thus, the server sends a confirmation to the client to allow it to connect to the SDN, or else sends the server to the controller to drop the device from the network if the device is unauthorized.

When the client sends the data to the authentication server, it is encrypted by Elliptic Curve Cryptography (ECC) to ensure the confidentiality of the data and not be sniping like a man-in-the-middle attack.

Although the products and standards that employ RSA for encryption and authentication are public-key, in recent years, the required key length for secure RSA use has increased.

ECC is more difficult to use than RSA. For high-level security functions such as encryption and authentication, ECC employs arithmetic algorithms as the primary objective operations. Both software and hardware can use ECC, which uses a standard to produce their public, and private

keys, and parties agree on publicly available data items. ECC's primary advantage over RSA is provided equivalent security for a much smaller key size, which lowers processing overhead. i.e., overall, ECC is more effective and secure than RSA [26].

To explain Elliptic Curve Cryptography (ECC) is based on the cubic equation:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

Eq. (1)

Where $a, b, c,$ and e are real integers and x and y take values in the real numbers, defines an elliptic curve over a finite field in two variables. The variables and coefficients for cryptography are limited to elements in a finite field that are binary curves over GF(2^m) and prime curves over Z_p. We utilize the cubic equation for a prime curve over Z_p [27].

$$y^2 \, mod \, p = (x^3 + ax + b) \, mod \, p$$

Eq. (2)

Where all the variables and coefficients have values in the range of integers from 0 to p-1 and where computations are done modulo p. Emphasizes that for software applications, prime curves are ideal. Values a=3, b= 1, and p = 3119 were offered from the curve that was used in Eq. (3).

$$y^2 \, mod \, 3119 = (x^3 + 3x + 1) \, mod \, 3119$$

Eq. (3)

We need to identify a "hard issue" that is the discrete logarithm or factoring the product of two primes to create an elliptic curve-based encryption system. Consider the formula:

$\mathcal{Q} = \mathcal{K}\mathcal{P}$, where$\mathcal{Q}$, $\mathcal{P} \in E_p(a, b)$, and $\mathcal{K}\mathcal{P}$ are all variables.

If $\mathcal{K}$ and$\mathcal{P}$ are known, it is quite simple to compute$\mathcal{Q}$, but, if $\mathcal{Q}$ and$\mathcal{P}$ are known, it is relatively challenging to find k which is known as the discrete logarithm problem for elliptic curves. The difficulty of calculating k using KP and P determines ECC security. The encryption and decryption system needs a point G and an ellipse set $E_p(a, b)$ as parameters. G Is a base point on the elliptic curve equation, generator point, denoted by the equation$G = (x_g , y_g)$. To explain the steps of encryption and decryption using ECC [28], it would be via our authentication approach that includes this encryption and decryption. This approach consists of three Python modules: a **server** module, a **client** module, and a **coding** module, which the pseudocode in Algorithm 2 and Figure 3 illustrates.

Step 1: In the beginning, this protocol is, as we said earlier, between the authentication server and clients to authenticate their devices, both the server and the client agree G and both select a private key where n_b and n_a for server and client, respectively. They compute the public key$\mathcal{P}$, which is the product of G and the private key, then they exchange public keys. The authentication server requests device data - the MAC address- from any client who connects to the network.

Step 2: The client module selects a random number and encrypts his device's MAC address using ECC before sending it to the server. The plaintext message (MAC address) must be encoded first as a (x , y) point Pm by calling the encode method in the coding module before being delivered. This method requires that the ASCII values of the characters be converted to their equivalent values. $\mathcal{K}$ = 20 is an auxiliary base parameter; both the client and the server should agree upon this. Receive all points on the elliptic curve that are used as (x, y)points.

If m is an ASCII value for each character, for each number $m\mathcal{K}$ take$x = m\mathcal{K} + 1$. By searching about x in P to find y that checks the **Eq. (3)**, try$x = m\mathcal{K} + 2$and then $x = m\mathcal{K} + 3$until one can

solve for y. In reality, one will encounter such a y before reaching$x = m\mathcal{K} + \mathcal{K} - 1$. Then take that point$(x, y)$. As a result, the value of m is now represented as a point on the elliptic curve. In this way, the entire argument is reduced to a series of points. Every pair will serve as a "P_m" input to the ECC mechanism.

Step 3: As for the encrypt step, in the **client** module, for each encoded text "P_m" as follows, it selects a random positive integer (k), then it computeskG and $\mathcal{P}_b$, where $\mathcal{P}_b$ is the public key of the server using point multiplication, then calls the encode method in the **coding** module to receive P_m then computes $P_m + k\mathcal{P}_b$ using point addition, and finally, it sends the ciphertext $\mathcal{C}_m$ to the server.

$$\mathcal{C}_m = \{kG, P_m + k\mathcal{P}_b\}$$

Eq. (4)

Step 4: In the **server** module, it decrypts each ciphertext as follows: it multiplies the first point in the pair $\mathcal{C}_m$ by the server's private key n_b and subtracts the result from the second point.

$$P_m + k\mathcal{P}_b - n_b\,(kG) = P_m + k\,(n_b\,G) - n_b\,(kG) = P_m$$

Eq. (5)

Then call the decodes method in the **coding** module to convert encoded P_m as point (x, y) to m (MAC address) to be the greatest integer less than $(x - 1)/k$ [29]. The client has added $k\mathcal{P}_b$ to the data to hide it. Although $\mathcal{P}_b$ is a public key, only the client knows the value of$k$, making it impossible to remove the mask$k\mathcal{P}_b$. The client does, however, moreover contain a "clue," which is sufficient to remove the mask if one is aware of the server's private keyn_b. Attackers would need to compute k by using G andkG, which is thought to be difficult, to recover the message.

Step 5**:** The **server** module has a file that contains the MAC addresses of the devices authorized to access the network (the auth file); it compares the client's MAC address with those addresses. If it matches one of them, it confirms that it will allow the client device to access the network. Otherwise, the server sends a message to the controller by calling the delete host method in the **reroute** module, which removes the unauthorized device from the network.

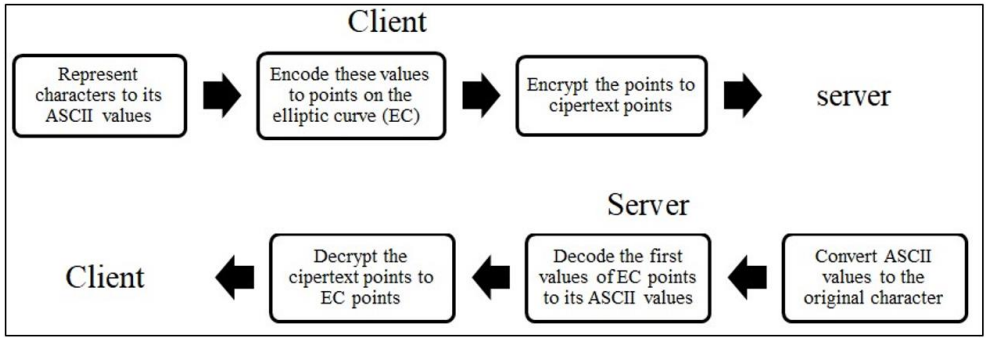

Figure 3. Flowchart for authentication approach

Algorithm 2

Client module

1- *Select* k, n_a
2- *Compute* $\mathcal{P}_a = n_a\, G$
3- *Send request Access to network* $\| \mathcal{P}_a$ *to server*
5- *Call encode method to convert MAC to* P_m *and get* P_m
6- *Encrypt* P_m *to* $C_m = \{kG, P_m + k\mathcal{P}_b\}$
7- *Send* C_m *to server*

Server module

1-*Select* n_b
2-*Compute* $\mathcal{P}_b = n_b\, G$
4-*Send replay request Access* $\| \mathcal{P}_b$ *to client*
8-*Decrypt cm obtain* $P_m = P_m + k\,(n_b\, G) - n_b\,(kG)$
9-*Call decode method to convert* P_m *to MAC and get MAC*
10- *Decrypt to obtain MAC*
11- *Compare MAC with MAC s in the auth file*

```
If MAC in file
   Confirm device for accessing to the network
Else If
   Send to Controller to drop device from network
End If
```

Coding module

encode method

```
{XX = [ ],Pm = [ ] and P = [ ]
Get all points P as (x,y) in list on elliptic curve
Plaintext = MAC address
M = list(Plaintext.encode('ascii'))
for jx in range(0,len(M)):
   for j in range(100):
      X = k * M[jx] + (j + 1)
          If X in firs element tuples of P get
          second element in tuples of P = Y1
          computeY2as(X ** 3 + a * X + b)%p
            If Y1 == Y2:
              Y1 = Y2 = Y
              output = (X,Y)
              XX.append(X)
              Pm.append(output)
          break
      else:
          continue}
```

decode method ()

```
{
 k = 20
 a = len(XX)
 for s in range(0,a):
    for X in XX:
       x = (X − 1)/k
       tex = round(x)
       n.append(tex)

m = ''.join(chr(i) for i in n)
 }
```

4. EXPERIMENT TESTING AND ANALYSIS RESULTS

4.1. Experiment Setup

The test supports the suggested proposal. On the Ubuntu 18 system, the SDN simulation environment using Ryu, Mininet, and the network topology is in Figure 2. The test on a computer with an Intel(R) Core(TM) i7-4600CPU 2.90 GHz CPU and 8 GB of RAM.

4.2. Experimental Results and Analysis

To receive results from the rerouting approach, use two terminals. One terminal to run the reroute module, and another terminal to run Mininet to simulate topology (Fattree topology), then run the pingall command in the Mininet terminal to test connectivity between all hosts. For example, Table 1, illustrated the paths from h1 to h4. Where h = host and s = switch, the algorithm selected the shortest path, Path 1 because it has fewer weights of edges than Path 2 (less in the number of hops); the link bandwidth utilization for Path 1 becomes 94.

Table 1. paths from $h1$ to $h4$

Paths	Edges (Switches)	The link bandwidth utilization
Path1	$[s6, s7]$	$(s6, s7) = 97 - 3 = 94$
Path2	$[s6, s2, s7]$	$(s6, s2) = 100$ $(s2, s7) = 100$

The higher the flows, the greater the consumption of link bandwidth thus if the link bandwidth consumption rate is less than the threshold of the link bandwidth utilization, then this link will be the congested link, such as some of the paths from $h7\ to\ h9$ in Table 2.

Table 2. paths from $h7$ to $h9$

Paths	Edges (Switches)	The link bandwidth utilization
Path1	$[s9, s3, s4, s10]$	$(s9, s3) = 73$ $(\boldsymbol{s3}, \boldsymbol{s4}) = \mathbf{1}$ $(s4, s10) = 61$
Path2	$[s9, s8, s3, s4, s\ 10]$	$(s9, s8) = 88$ $(s8, s3) = 28$ $(\boldsymbol{s3}, \boldsymbol{s4}) = \mathbf{1}$ $(s4, s10) = 61$
Path3	$[s9, s3, s1, s4, s10]$	$(s9, s3) = 73$ $(s3, s1) = 100$ $(s1, s4) = 100$ $(s4, s10) = 61$
Path4	$[s9, s3, s4, s11, s10]$	, $(s4, s11) = 64$ $(s11, s10) = 100$ $(s9, s3) = 73$ $(\boldsymbol{s3}\ \boldsymbol{s4})\quad \mathbf{1}$

The algorithm selected the shortest path, path1 because it has fewer weights of edges than other paths, thus the link bandwidth utilization for Path1 is in Table 2. But, the algorithm detected congestion in this path by computing rem_bw for a link $(s3, s4)$ in the path that is less than BW, thus it is a congested link.

The algorithm reroutes to another link that has the shortest path that is path 3 which has rem_bw less than BW, unlike other paths in Table 2.

The network bandwidth measuring tool and traffic generator both use the $\boldsymbol{iperf}$ tool **[30]**. We use $\boldsymbol{iperf}$ to measure throughput between $h4$ $h10$ in TCP flow by run" $\boldsymbol{xterm}$ $h2$ $h10$" in Mininet terminal, start the TCP server $(-s)$ at $h1$ with port 5566 $(-p)$. Moreover, monitor the results every second $(-i)$ to run in h1 terminal "$\boldsymbol{iperf}$ $-s -p$ 5566 $-i$ 1". Start the TCP client (–c) at h10. Set the transmission duration (-t) to 100 seconds after -c, one need to specify the server IP address 10.0.0.4 to run in h1 terminal "$\boldsymbol{iperf}$–c 10.0.0.2 –p 5566 –t 100".

By running the Mininet emulator in one terminal and the Ryu controller's **reroute module** in another terminal, then typing "$h2$ ping $h10$" in the Mininet terminal, the round trip time (RTT) of the network can be measured since it has a significant impact on determining the speed and dependability of the network connection. To receive results from the authentication approach, run $\boldsymbol{xterm}$ in Mininet and, using the Ryu controller's **sdn_reroute** application for all hosts, apply the device authentication algorithm by running the **server module** in $\boldsymbol{xterm}$ $h1$ and the **client module** in another $\boldsymbol{xterm}$.

The **server** module confirms hosts or devices can access the network except for unauthorized devices; refer to the auth file. In our algorithm, select h3, h9, h6, and h13 as unauthorized devices that were dropped by the Ryu controller. We use $\boldsymbol{iperf}$ to measure throughput between $h4$ and $h10$ again after applying the authentication approach in the same steps in the rerouting approach. In Figure 4 TCP Flow with Authentication, we use $\boldsymbol{iperf}$ to measure throughput between $h4$ and $h10$ by running "$\boldsymbol{xterm}$ $h4$ $h10$" in Mininet Terminal, following the same steps as in the first stage of the congestion control mechanism. By comparing them, we find that the throughput has improved when we apply the device authentication algorithm.

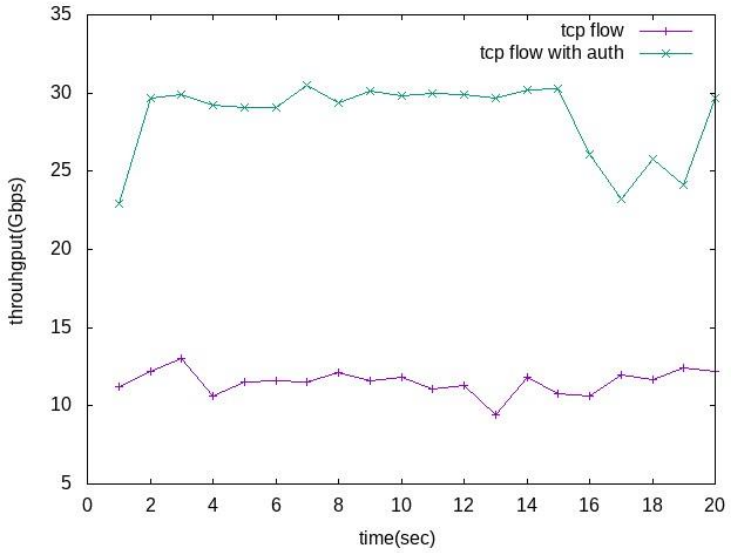

Figure 4. The throughputs between $\boldsymbol{h4}$ $\boldsymbol{and}$ $\boldsymbol{h10}$

The round-trip time (RTT) of the network can be measured again after the device authentication approach, as in the same steps in the congestion control approach.

In Figure 5, the round trip time between $h2$ and $h10$ is illustrated in the congestion control mechanism and after the device authentication algorithm. By comparing it, we notice that the RTT

is lower after the device authentication algorithm is applied, as the minimum RTT decreases from 0.119 to $0.050ms$.

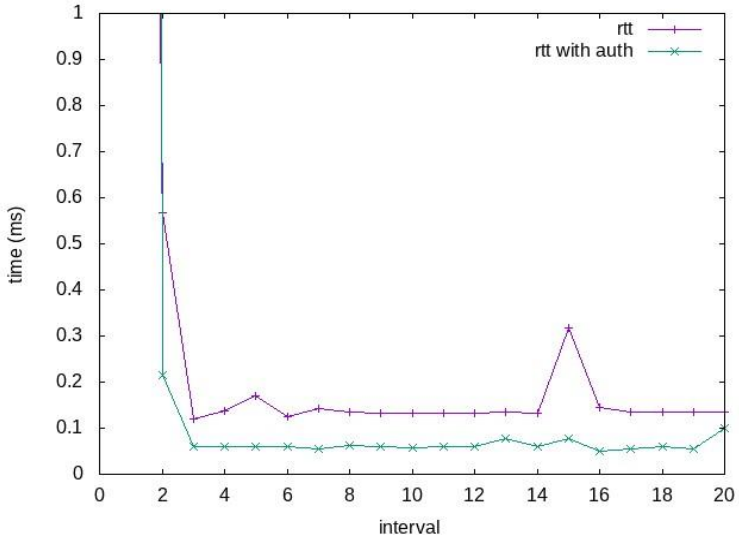

Figure 5. The round trip time between $\boldsymbol{h2}$ and $\boldsymbol{h10}$

5. CONCLUSION

To improve network performance and ensure the quality of the SDN service, the first part of our thesis computes all shortest paths between any two edges of an SDN network using a Ryu controller and Mininet. The second part of our thesis controls congestion by identifying the congested link with the lowest bandwidth and then choosing the shortest path with higher bandwidth than the link's maximum capacity to change the flow path. A device authentication mechanism is then added between client devices and the authentication server to prevent unauthorized devices from accessing the network by the controller, increase throughput, and decrease round-trip times.

REFERENCES

[1] M. R. Belgaum, S. Musa, M. M. Alam, and M. M. Su'ud, "A systematic review of load balancing techniques in software-defined networking," *IEEE Access*, vol. 8, pp. 98612-98636, 2020.

[2] J. E. T. Regencia and W. E. S. Yu, "Latency and throughput advantage of leaf-enforced quality of service in software-defined networking for large traffic flows," in *Intelligent Computing*: Springer, 2021, pp. 606623.

[3] A. Shirmarz and A. Ghaffari, "Performance issues and solutions in SDN-based data center: a survey," *The Journal of Supercomputing*, vol. 76, no. 10, pp. 7545-7593, 2020.

[4] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, "A comparison between several Software Defined Networking controllers," in *2015 12th international conference on telecommunication in modern satellite, cable and broadcasting services (TELSIKS)*, 2015: IEEE, pp. 223226.

[5] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, and M. Keshtgary, "A survey on SDN, the future of networking," *Journal of Advanced Computer Science & Technology*, vol. 3, no. 2, pp. 232-248, 2014.

[6] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadiq, and S. Mounir, "A comprehensive survey on SDN security: threats, mitigations, and future directions," *Journal of Reliable Intelligent Environments*, pp. 1-39, 2022.

[7] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Interactions between congestion control algorithms," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019: IEEE, pp. 161-168.

[8] J. Zhang, Z. Yao, Y. Tu, and Y. Chen, "A survey of TCP congestion control algorithm," in *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*, 2020: IEEE, pp. 828-832.

[9] Z. Gál, G. Kocsis, T. Tajti, and R. Tornai, "Performance evaluation of massively parallel and high speed connectionless vs. connection-oriented communication sessions," *Advances in Engineering Software*, vol. 157, p. 103010, 2021.

[10] G. Kumar *et al.*, "Swift: Delay is simple and effective for congestion control in the data center," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 514-528.

[11] L. P. Verma and M. Kumar, "An IoT-based congestion control algorithm," *Internet of Things*, vol. 9, p. 100157, 2020.

[12] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (SDN) data plane security: issues, solutions, and future directions," *Handbook of Computer Networks and Cyber Security*, pp. 341387, 2020.

[13] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in SDN: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[14] S. N. Hertiana and A. Kurniawan, "A joint approach to multipath routing and rate adaptation for congestion control in OpenFlow Software Defined Network," in *2015 1st International Conference on Wireless and Telematics (ICWT)*, 2015: IEEE, pp. 1-6.

[15] R. Kanagevlu and K. M. M. Aung, "SDN controlled local re-routing to reduce congestion in cloud data center," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, 2015: IEEE, pp. 80-88.

[16] M. Gholami and B. Akbari, "Congestion Control in Software Defined Data Center Networks Through Flow Rerouting," in *2015 23rd Iranian conference on electrical engineering*, 2015: IEEE, pp. 654-657.

[17] U. Zakia and H. B. Yedder, "Dynamic load balancing in SDN-based data center networks," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2017: IEEE, pp. 242-247.

[18] D. M. Ferrazani Mattos and O. C. M. B. Duarte, "AuthFlow: authentication and access control mechanism for software defined networking," *annals of telecommunications*, vol. 71, no. 11, pp. 607-615, 2016.

[19] J. Liu, Y. Lai, Z. Diao, and Y. Chen, "A trusted access method in software-defined network," *Simulation Modelling Practice and Theory*, vol. 74, pp. 28-45, 2017.

[20] S. Bhardwaj and S. N. Panda, "Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment," *Wireless Personal Communications*, vol. 122, no. 1, pp. 701-723, 2022.

[21] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.

[22] D. Dholakiya, T. Kshirsagar, and A. Nayak, "Survey of Mininet Challenges, Opportunities, and Application in Software-Defined Network (SDN)," in *International Conference on Information and Communication Technology for Intelligent Systems*, 2020: Springer, pp. 213-221.

[23] O. Oginni, P. Bull, and Y. Wang, "Constraint-aware software-defined network for routing real-time multimedia," *ACM SIGBED Review*, vol. 15, no. 3, pp. 37-42, 2018.

[24] P. Wills and F. G. Meyer, "Metrics for graph comparison: a practitioner's guide," *Plos one*, vol. 15, no. 2, p. e0228728, 2020.

[25] K. Nisar *et al.*, "A survey on the architecture, application, and security of software defined networking: Challenges and open issues," *Internet of Things*, vol. 12, p. 100289, 2020.

[26] F. Mallouli, A. Hellal, N. S. Saeed, and F. A. Alzahrani, "A survey on cryptography: comparative study between RSA vs ECC algorithms, and RSA vs El-Gamal algorithms," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2019: IEEE, pp. 173-176.

[27] W. Stallings, "Cryptography and Network Security Principles and Practice Seventh Edition Global Edition British Library Cataloguing-in-Publication Data," 2017.

[28] A. Srivastava and A. Kumar, "A review on authentication protocol and ECC in IoT," in *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, 2021: IEEE, pp. 312-319.

[29] P. Bh, D. Chandravathi, and P. P. Roja, "Encoding and decoding of a message in the implementation of Elliptic Curve cryptography using Koblitz's method," *International Journal on Computer Science and Engineering*, vol. 2, no. 5, pp. 1904-1907, 2010.

[30] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "iPerf-The ultimate speed test tool for TCP, UDP and SCTP," *línea]. Available:* https://iperf. *fr.[Último acceso: 23 Mayo 2018]*, 2019.